高等学校培养应用型人才教材——计算机系列

数据结构（C语言版）

张家超　主编
滕敏　刘跃建　副主编
王学卿　主审

中国电力出版社
www.infopower.com.cn

高等学校培养应用型人才教材——计算机系列

数据结构（C语言版）

张家超　主编
滕敏　刘跃建　副主编
王学卿　主审

中国电力出版社

内 容 提 要

本书从技术的先进性、实用性出发，分别介绍了数据结构的基本概念、数组和矩阵、线性表、栈与队列、树和二叉树、图、排序、查找等有关概念、基本运算及重要算法。为帮助深入理解课程教学内容，本书最后还提供了上机实验和课程设计指导，通过上机实践可以逐步掌握程序设计的基本过程、数据结构的应用和算法的实现，为应用软件的开发打下良好的实践基础。

本书不仅可以作为应用型高等院校讲授数据结构知识的参考教材，亦可以作为应考计算机软件人员水平（资格）考试“数据结构”部分的参考资料。

图书在版编目（CIP）数据

数据结构（C 语言版）/ 张家超主编. —北京：中国电力出版社，2003

高等学校培养应用型人才教材——计算机系列

ISBN 7-5083-1527-8

Ⅰ.数... Ⅱ.张... Ⅲ.①数据结构－高等学校－教材 ②C 语言－程序设计－高等学校－教材 Ⅳ.①TP311.12 ②TP312

中国版本图书馆 CIP 数据核字（2003）第 115739 号

中国电力出版社出版、发行

（北京三里河路 6 号 邮政编码 100044）

汇鑫印务有限公司印刷

*

2004 年 2 月第一版 2004 年 2 月北京第一次印刷

787 毫米×1092 毫米 16 开本 12 印张 261 千字

印数 0001－4000 册 定价 **18.00** 元

高等学校培养应用型人才教材——计算机系列

编　委　会

序　言

进入21世纪，世界高等教育已从精英教育走向了大众教育。我国也适应这一潮流，将高等教育逐步推向大众化。培养应用型人才已成为国家培养国际人才的重要组成部分，且得到了社会各界的广泛支持。于是一大批有规模、有实力、规范化、以培养应用型人才为己任的高等学校得到了长足发展。这类高校办学的一个显著的特点是按照新时代需求和当地的需要来培养学生，他们重视产学研相结合，并紧密地结合当地经济状况，把为当地培养应用型人才作为学校办学的主攻方向。

这类学校的教学特点是：在教授“理论与技术”时，更注重技术方法的教学。在教授“理论与实践”时，更注重理论指导下的可操作性，更注意实际问题的解决。因此，这些学生善于解决生产中的实际问题，受到地方企事业单位的普遍欢迎。

为满足这类高校的教学要求，达到培养应用型人才的目的，根据教育部有关重点建设项目的要求和相关教学大纲，我们组织了多年在这类高校中从教，并具有丰富工程经验的资深教授、高级工程师、教师来编写这套教材。

在这套教材的编写中，我们提倡“实用、适用、先进”的编写原则和“通俗、精练、可操作”的编写风格，以解决多年来在教材中存在的过深、过高且偏离实际的问题。

实用——本套教材重点讲述本行业中最广泛应用的知识、方法和技能。使学生学习后能胜任岗位工作，切实符合当地经济建设的需要和社会需要。

适用——本套教材是以工程技术为主的教材，所以它适用于培养应用型人才的所有高校（包括本科、专科、技术学院、高职等），既符合此类学生的培养目标，又便于教师因材施教。

先进——本套教材所选的内容是当今的新技术、新方法。使学生在掌握经典的技术和方法之后，可用教材中的新技术、新方法去解决工程中的技术难题，为学生毕业后直接进入生产第一线打下坚实的基础。

通俗——本套教材语言流畅、深入浅出、容易读懂。尽量避开艰深的理论和长篇的数学推导，尽量以实例来说明问题，在应用实例中掌握理论，使学生轻松掌握所学知识技能，达到事半功倍的效果。

精练——本套教材选材精练、详细而不冗长、简略得当，对泛泛而谈的内容将一带而过，对学生必须掌握的新技术、新方法详细讲，讲透、讲到位，为教师创造良好的教学空间和结合当地情况调整教学内容的余地。

可操作——本套教材所有的实例均是容易操作的，且是有实际意义的案例。把这些案例连接起来，就是一个应用工程的实例。通过举一反三的应用，使学生能够在更高层次上创造性地应用教材中的新思想、新技术、新方法去解决问题。

本套教材面向培养应用型人才的高等学校，同时也可作为社会培训高级技术人才的教材和需要加深某些方面知识技能的人员的自学教材。

编　委　会

前　言

《数据结构》是高等院校计算机及相关专业的一门重要的、经典的专业基础课程，也是核心课程之一。计算机软硬件的飞速发展，带给高等院校师生的不仅仅是技术上的，也是观念上的快速变革。在教学第一线的教师们，经过多年的探索，结合市场的需求，以及计算机软件人员水平（资格）考试的内容，编写了本书。

本书中所有的算法都采用C语言编写，程序中做了必要的注释，并且都在机器上调试成功；即使是程序片段（或函数），只要加上必要的、简单的调用函数，也都可以很容易地调试成功。

本书不仅可以作为应用型高等院校讲授数据结构知识的参考教材，亦可以作为应考计算机软件人员水平（资格）考试“数据结构”部分与算法设计部分的参考资料。

本书共分9章。第1章介绍了数据结构的基本概念，第2章介绍了数组和矩阵，第3章介绍了线性表，第4章介绍了栈与队列，第5章介绍了树和二叉树，第6章介绍了图，第7章介绍了排序，第8章介绍了查找，为帮助深入理解课程教学内容，本书第9章还提供了上机实验和课程设计指导，通过上机实践可以逐步掌握程序设计的基本过程，数据结构的应用和算法的实现，为应用软件的开发打下良好的实践基础。

本书由张家超主编，滕敏、刘跃建副主编，第1章由张家超编写，第2、3章、第9章课程设计指导部分由刘跃建编写，第4章和第8章由朱翠苗编写，第5、6章由滕敏编写，第7章由刘丽编写，第9章上机实验指导部分由任中林编写，全书由张家超负责统稿、定稿，王学卿主审。

本书在编写过程中，还得到了许多高校同行们的大力支持和帮助，参考了许多已经和未经出版的教材、讲义等，在此不一一列举，非敢掠美；出版时也得到了中国电力出版社的大力帮助，没有他们热心的支持和辛勤的劳动，本书的出版是不可能的，在此，一并表示衷心的感谢。

由于编著者水平及时间有限，书中错漏和不妥之处在所难免，恳请专家和读者批评、指正。

目　录

第 1 章　数据结构概论

1.1　数据结构的基本概念

计算机科学是一门研究数据和数据处理的科学。数据是计算机化的信息，它是计算机可以直接处理的最基本和最重要的对象。无论是进行科学计算或数据处理、过程控制以及对文件的存储和检索及数据库技术等计算机应用领域中，都是对数据进行加工处理的过程。因此，要设计出一个结构好效率高的程序，必须研究数据的特性及数据间的相互关系及其对应的存储表示，并利用这些特性和关系设计出相应的算法和程序。

数据结构是计算机科学与技术专业的专业基础课，是十分重要的核心课程。所有的计算机系统软件和应用软件都要用到各种类型的数据结构。因此，要想更好地运用计算机来解决实际问题，仅掌握几种计算机程序设计语言是难以应付众多复杂的问题的。要想有效地使用计算机、充分发挥计算机的性能，还必须学习和掌握好数据结构的有关知识。打好“数据结构”这门课程的扎实基础，对于学习计算机专业的其他课程，如操作系统、编译原理、数据库系统、软件工程、人工智能等都是十分有益的。

1.1.1　常用术语

1. 数据（Data）

是对客观事物的符号表示，在计算机科学中是指所有能输入到计算机中并被计算机程序处理的符号的总称。它是计算机程序加工的“原料”。数据的含义极为广泛，如图形、图像、声音等都可以通过编码而归之于数据的范畴。

2. 数据元素（Data Element）

是数据的基本单位，在计算机程序中通常作为一个整体进行考虑处理。例如：一个班级学生的成绩中单个学生成绩为一个数据元素。一个数据元素可由若干个数据项组成，如单个学生成绩中各门课程的成绩为单个数据项。数据项是数据的不可分割的最小单位。

3. 数据对象（Data Object）

是性质相同的数据元素的集合，是数据的一个子集。例如，字母字符数据对象是集合 C={ ‘A’, ‘B’, …, ‘Z’ }。

4. 数据类型（Data Type）

数据类型是和数据结构密切相关的一个概念。它最早出现在高级程序设计语言中，用以刻画程序中操作对象的特性。在用高级语言编写的程序中，每个变量、常量或表达式都有一个它所属的确定的数据类型。类型显式地或隐含地规定了在程序执行期间变量或表达式所有可能的取值范围，以及在这些值上允许进行的操作。因此，数据类型（Data Type）是一个值的集合和定义在这个值集上的一组操作的总称。

在高级程序设计语言中，数据类型可分为两类：一类是原子类型；另一类则是结构类型。原子类型的值是不可分解的。如 C 语言中整型、字符型、浮点型、双精度型等基本类型，分别用保留字 int、char、float、double 标识。而结构类型的值是由若干成分按某种结构组成的，因此是可分解的，并且它的成分可以是非结构的，也可以是结构的。例如，数组的值由若干分量组成，每个分量可以是整数，也可以是数组等。在某种意义上，数据结构可以看成是“一组具有相同结构的值”，而数据类型则可被看成是由一种数据结构和定义在其上的一组操作所组成的。

5. 抽象数据类型（Abstract Data Type，ADT）

指一个数学模型以及定义在该模型上的一组操作。抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。可用以下三元组表示：（D，S，P），其中 D 是数据对象；S 是 D 上的关系集；P 是对 D 的基本操作集。

1.1.2 数据的结构

1. 数据结构

数据结构是在整个计算机科学与技术领域广泛被使用的术语。它用来反映一个数据的内部构成，即一个数据由哪些成分数据构成，以什么方式构成，呈什么结构。数据结构有逻辑上的数据结构和物理上的数据结构之分。逻辑上的数据结构反映成分数据之间的逻辑关系，而物理上的数据结构反映成分数据在计算机内部的存储安排。数据结构是数据存在的形式。

数据结构是信息的一种组织方式，其目的是为了提高算法的效率，它通常与一组算法的集合相对应，通过这组算法集合可以对数据结构中的数据进行某种操作。

数据结构作为一门学科主要研究数据的各种逻辑结构和存储结构，以及对数据的各种操作。因此，主要有三个方面的内容：数据的逻辑结构；数据的物理存储结构；对数据的操作（或算法）。通常，算法的设计取决于数据的逻辑结构，算法的实现取决于数据的物理存储结构。

数据结构是相互之间存在一种或多种特定逻辑关系的、按照一定的存储表示方式存储

在计算机的存储器中、并定义了一个运算集的一组数据元素的集合。根据数据元素之间关系的不同，通常有下列四类基本结构，如图 1-1 所示。

（1）集合结构：松散关系。

（2）线性结构：结构中的数据元素为一对一的关系。

（3）树形结构：结构中的数据元素为一对多的关系。

（4）图状结构或网状结构：结构中的数据元素为多对多的关系。

数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构可以看做是从具体问题抽象出来的数学模型，它与数据的存储无关。我们研究数据结构的目的是为了在计算机中实现对它的操作，为此还需要研究如何在计算机中表示一个数据结构。数据结构在计算机中的标识（又称映像）称为数据的物理结构，或称存储结构。它所研究的是数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。

我们经常称这样一种数据结构为文件，其中的数据元素为记录（又称为结点），而记录又可以有不可再分的数据项（又称为字段）组成，这些数据项中有一个或多个数据项可以惟一地标识一个记录，这样的数据项称为关键字（key）。

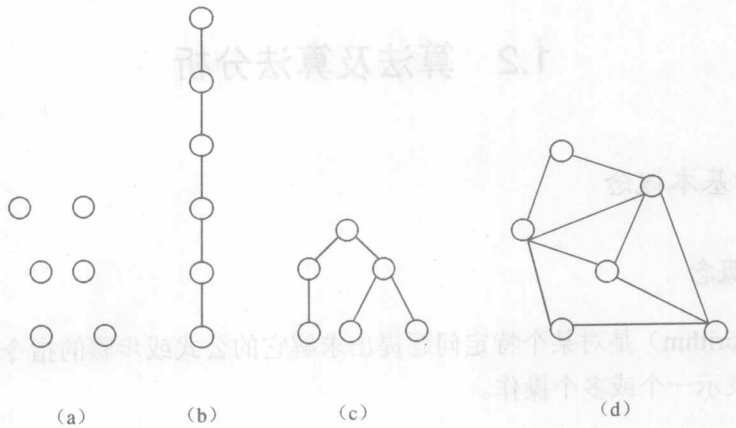

图 1-1　几种常见的数据的结构

（a）集合结构；（b）线性结构；（c）树形结构；（d）网状结构

2. 数据的存储结构

数据的存储结构可采用以下四种存储方法来实现。

（1）顺序存储方法：是把逻辑上相邻的元素存储在物理位置相邻的存储单元中，由此得到的存储表示称为顺序存储结构。顺序存储结构是一种最基本的存储表示方法，数据元素之间的关系是由存储单元的邻接关系来体现的，在高级程序设计语言中通常用数组来实现。

（2）链式存储方法：是对逻辑上相邻的元素不要求其物理位置相邻，元素间的逻辑关系通过附设的指针字段来表示，由此得到的存储表示称为链式存储结构，链式存储结构在

高级程序设计语言中通常用指针类型来实现。

显然，链式存储结构额外增加了一个称为指针字段的存储空间，因此称之为非紧凑结构，而顺序存储结构的存储空间全部分配给了数据元素，所以称之为紧凑结构。

定义结构的存储密度为数据本身所占的存储量与整个结构所占存储量之比。顺序存储结构的存储密度为 1，链式存储结构的存储密度小于 1。存储密度越接近于 1，存储空间的利用率越大。当然，经过合理设计的附加信息会给运算带来很大的方便，如链式存储中的指针字段，在数据元素的插入、删除等运算方面要比顺序存储结构方便得多。这种以空间换时间（反之亦然）的方法，在算法设计过程中是经常要进行选择的。

（3）索引存储方法。除了数据本身之外，另外再建立一张指示逻辑数据与物理数据之间一一对应关系的表，即索引表。

（4）散列存储方法。主要思想是根据结点的值来确定存储地址，即结点的存储地址是结点值的函数，该函数称为散列函数。

一般的数据结构的存储都是这四种存储方法之一，或是它们的组合。同一个逻辑结构可以有几种不同的存储方式，选择哪一种要根据具体的运算要求来确定。

1.2 算法及算法分析

1.2.1 算法的基本概念

1. 算法的概念

算法（Algorithm）是对某个特定问题提出求解它的公式或步骤的指令的有限序列，其中每一个指令表示一个或多个操作。

2. 算法的特性

（1）有穷性：一个算法必须总是（对任何合法的输入值）在执行有穷步之后结束，且每一步都可在有穷时间内完成。当然某些需要强制执行的系统如操作系统等除外。

```
haha()
{  /* only a joke, do nothing. */  }
main()
{  printf("请稍等…您将知道世界的末日…");
   while(1) haha();
}
```

（2）确定性：算法中每一条指令必须有确切的含义，读者理解时不会产生二义性。在任何条件下，算法只有惟一的一条执行路径，即对于相同的输入只能得出相同的输出。算

法执行完毕后，可以得到问题的解或指出问题无解。

```
float average(int *a,int num)
{ int i;long sum=0;
   for(i=0;i<num;i++)
   sum+=*(a++);
   return sum/num;
}
main()
{  int score[10]={1,2,3,4,5,6,7,8,9,0};
   printf("%f", average(score,10);
}
```

（3）可行性：一个算法是能行的，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

（4）输入：一个算法有零个或多个的输入，这些输入取自于某个特定的对象的集合。

（5）输出：一个算法有一个或多个的输出，这些输出是同输入有着某些特定关系的量。

```
getsum(int num)
{   int i,sum=0;
    for(i=1;i<=num;i++)
    sum+=i;
} /* 无输出的算法没有任何意义 */
```

3. 算法设计要求

（1）易读性：算法主要是为了人的阅读与交流，其次才是机器执行。易读性好有助于对算法的理解。

（2）正确性：有正确结果输出。

程序不含语法错误。程序对于几组输入数据能够得出满足规格说明要求的结果。程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出满足规格说明要求的结果。程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

（3）健壮性：不同的输入有相应的响应。

（4）效率和存储量的要求：时间和空间复杂性。

效率指的是算法执行时间。对于解决同一问题的多个算法，执行时间短的算法效率高。存储量需求指算法执行过程中所需要的最大存储空间。两者都与问题的规模有关。

【例 1-1】在三个整数中求最大者。

算法 1-1　在三个整数中求最大者。

```
max(int a, int b, int c)
```

```
{  if (a>b)
   {   if(a>c) return a;
       else return c;
   }
  else
   {   if(b>c) return  b;
       else      return  c;
}/* 无需额外存储空间，只需两次比较 */
```

算法 1-2　在三个整数中求最大者。

```
max(int a[3])
{  int c,int i;
   c=a[0];
   for(i=1;i<3;i++)
      if (a[i]>c) c=a[i];
   return c;
}
/* 需要两个额外的存储空间，两次比较，至少一次赋值 */
/* 共需 5 个整型数空间 */
```

4. 算法的分类

我们目前所说的算法主要有两大类：一类是数值算法；一类是非数值算法。

（1）数值计算算法。

例如，求方程 $Ax^2 + Bx + C = 0$ 的根，矩阵运算，可视化计算等，这些问题都有经典的算法，许多软件公司开发了很多软件包，以方便人们使用。这主要与科学计算应用计算机时间较长有密切关系。

（2）非数值计算算法。

例如，对一组数据进行排序，查找其中的某个数，在其中插入一个新数等。非数值计算问题，在管理科学、人工智能等领域中应用比较广泛，其中的许多问题没有现成的求解公式，需要人们经过长期的经验积累才能求出趋向于合理的结论。

1.2.2　算法的描述

算法可以使用各种不同的方法来描述。

1. 自然语言

用我们熟悉的自然语言（英语或汉语）来表达求解问题的过程（即算法），优点是简单直观，且便于人们对算法的阅读。当然自然语言表达的算法，缺点是不够严谨，与计算

机的具体的高级程序设计语言形式相差很大，需要读者认真地进行转换，反复练习，才能够掌握好。

例如：求 1 到 100 的和。

2. 数学公式

用数学公式表达算法主要适应于数值计算的问题，表达严谨，但局限性比较严重，而且同自然语言一样，与具体的高级程序设计语言形式相差较大。

例如：$\text{sum}=\sum_{i=1}^{100} i$

3. 流程图

使用程序流程图（如图 1-2 所示）、N-S 图（如图 1-3 所示）等算法描述工具。其特点是描述过程简捷、明了，比较容易转换成具体的高级程序设计语言的形式。

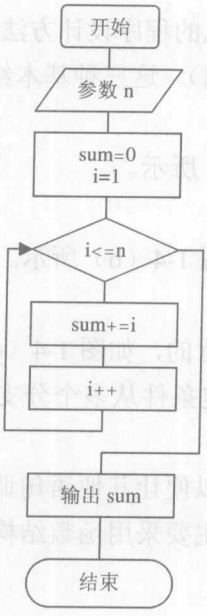

图 1-2　流程图

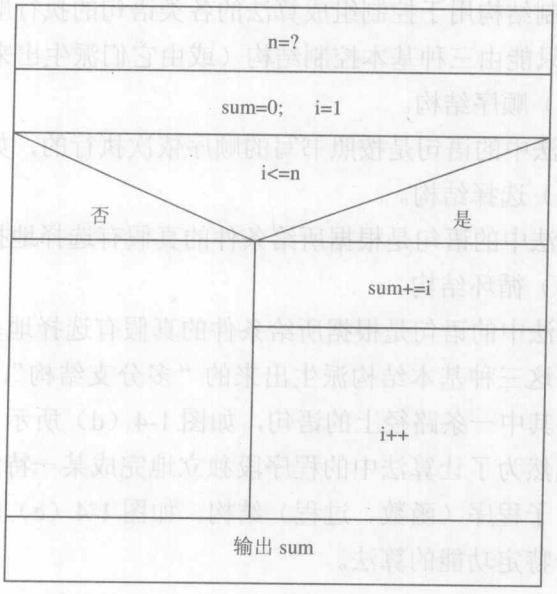

图 1-3　N-S 图

4. 伪代码

伪代码语言介于高级程序设计语言和自然语言之间，它忽略高级程序设计语言中一些严格的语法规则与描述细节，因此它比程序设计语言更容易描述和被人理解，而比自然语言更接近程序设计语言。它虽然不能直接执行但很容易被转换成高级语言。

（1）给 i 和 sum 赋初值 0；给 n 设定初值 100；

（2）while (i<=n) { sum = sum + i； i++ }；

（3）输出 sum。

1.2.3 算法的要素

算法的要素有两个：一个是操作；一个是控制结构。

1. 操作

构成算法的操作主要与程序设计语言有关，在高级程序设计语言中所描述的操作（一般称为元操作）主要包括：算术运算（+、-、*、/）、关系运算（>、=、<、>=、<=、==、!=）、逻辑运算（与、或、非）、函数运算、位运算、I/O 操作等。

数据结构所讨论的操作主要是指由以上这些操作经各种组合而成的复合操作，如更新（插入、删除、修改）、查询（查找）、排序等。

2. 控制结构

控制结构用于控制组成算法的各类语句的执行顺序。结构化的程序设计方法规定，一个程序只能由三种基本控制结构（或由它们派生出来的结构组成）。这三种基本结构是：

（1）顺序结构。

算法中的语句是按照书写的顺序依次执行的，如图 1-4（a）所示。

（2）选择结构。

算法中的语句是根据所给条件的真假有选择地执行的，如图 1-4（b）所示。

（3）循环结构。

算法中的语句是根据所给条件的真假有选择地多次重复执行的，如图 1-4（c）所示。

由这三种基本结构派生出来的“多分支结构”，即根据给定条件从多个分支路径中选择执行其中一条路径上的语句，如图 1-4（d）所示。

当然为了让算法中的程序段独立地完成某一特定的功能，以便让其他语句调用，从而构成了子程序（函数、过程）结构，如图 1-4（e）所示，本书主要采用函数结构来表达完成某一特定功能的算法。

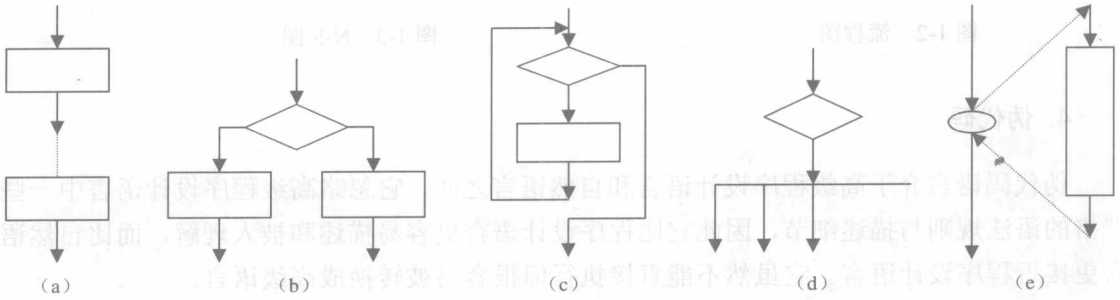

图 1-4 算法的各种结构

（a）顺序结构；（b）选择结构；（c）循环结构；（d）多分支结构；（e）子程序调用

1.2.4 算法的分析

算法执行的时间是算法优劣和问题规模的函数。评价一个算法的优劣，可以在相同的规模下，考察算法执行时间的长短来进行判断。而一个程序的执行时间通常有两种方法：

1. 事后统计

根据算法执行后的结果来分析算法的优劣。主要是利用计算机内记时的功能，将不同算法编写的程序，输入一组或多组相同的统计数据，来区分所获得的不同结果。缺点：一是必须先运行依据算法编制的程序；二是所得时间统计量依赖于硬件、软件等环境因素，掩盖算法本身的优劣，不利于较大范围内（异地、异时、异境）的算法比较。

2. 事前分析

某种高级语言编写的程序在计算机上运行所消耗的时间取决于：

- 依据的算法选用何种策略
- 问题的规模（规模越大，消耗时间越多）
- 程序语言（语言越高级，消耗时间越多）
- 编译程序产生机器代码质量
- 机器执行指令速度等硬件平台

所以使用绝对时间单位来衡量算法的效率是不完全合适的。因此，我们选用下面的手段来度量算法的效率。

（1）时间复杂性。

一个程序的时间复杂度（Time Complexity）是指程序运行从开始到结束所需要的时间。常用 T(n)=O(f(n))来表示。其中，f(n)为问题规模 n 的函数，一般用语句的执行次数来代替，O(x)读作“order of x”或大 O(Big-Oh)。

【例 1-2】求 4*4 矩阵元素和。

```
sum(int num[4][4])
{ int i,j,r=0;
   for(i=0;i<4;i++)
     for(j=0;j<4;j++)
       r+=num[i][j];       /* 原操作 */
   return r;
}
T(4)=O(f(4))
f=n*n

ispass(int num[4][4])
```